Chapter 2

The Programming Thing

In This Chapter

- ▶ Understanding programming history
- ▶ Creating source code
- ▶ Compiling source code into object code
- ▶ Using the Linker to create a program
- ▶ Testing the final program

It's called *programming,* though the cool kids know it as *coding* — the process whereby a human being writes information, sometimes resembling cryptic English, that is then somehow translated into directions for an electronic gizmo. In the end, this silent and solitary art grants individuals the power to control electronics. It's a big deal. It's the program thing.

The History of Programming

Few books written about programming get away with not describing the thrill-a-minute drama of programming history. As a programmer myself, it's difficult not to write about it, let alone contain my enthusiasm at cocktail parties. So consider this section optional reading, although a review of where programming has been and where it is today may help you better understand the art form.

In a nutshell, *programming* is the process of telling a gizmo what to do. That gizmo is *hardware;* the program is *software.*

Reviewing early programming history

The first gizmo to be programmed was Charles Babbage's analytical engine, back in 1822. The programming took place by physically changing the values represented by a column of gears. The engine would then compute the result of some dull, complex mathematical equation.

In the 1940s, early electronic computers were programmed in a similar manner to Babbage's analytical engine. A major difference was that rather than rearrange physical gears, instructions were hard-wired directly into electric circuitry. "Programming" pretty much meant "rewiring."

Over time, the rewiring job was replaced by rows of switches. Computer instructions were input by throwing switches in a certain way.

Professor John von Neumann pioneered the modern method of computer programming in the 1950s. He introduced decision-making into the process, where computers could make if-then choices. Professor von Neumann also developed the concept of the repeating loop and the subroutine.

It was Admiral Grace Hopper who developed the *compiler,* or a program that creates other programs. Her compiler would take words and phrases in English and translate them into computer code. Thus the programming language was born.

The first significant programming language was FORTRAN, back in the 1950s. Its name came from *for*mula *tran*slator. Other programming languages of the period were COBOL, Algol, Pascal, and BASIC, among others.

REMEMBER

Regardless of the form, whether it's rewiring circuits, flipping switches, or writing a programming language, the end result is the same: telling hardware to do something.

Introducing the C language

The C language was developed in 1972 at AT&T Bell Labs by Dennis Ritchie. It combined features from the B and BCPL programming languages but also mixed in a bit of the Pascal language. Mr. Ritchie, along with Brian Kernighan, used C to create the Unix operating system. A C compiler has been part of that operating system ever since.

In the early 1980s, Bjarne Stoustroup used C as the basis of the object-oriented C++ programming language. The ++ (plus-plus) part of the name is kind of an in-joke, which you'll understand better after reading Chapter 11, but

Mr. Stoustroup intended C++ to be the successor to C. In many ways it is, yet C remains popular.

The D programming language was introduced in the early 2000s. It's not as popular as C++, and it's only visually similar to C. Still, with the name *D,* the implication is that it's the next language after C. Or in this case, after C++.

- The B programming language, upon which C is based, was named after the *B* in Bell Labs.
- BCPL stands for Basic Combined Programming Language.
- The C++ programming language is quite similar to C, but it's not merely an extension or an add-on. It's easier to learn C++ when you know C, but it's not easy to switch between the languages.
- Unfortunately, I have no idea how to pronounce "Bjarne Stoustroup."

The Programming Process

No matter what you program or which language you use, certain procedures are common to the process. This section provides an overview.

Understanding programming

The goal of programming is to choose a language and utilize various tools to create a program. The language is C, and the tools are the editor, compiler, and linker — or an IDE, which combines all three. The end result is a program that directs the hardware to do something. That hardware can be a computer, tablet, phone, microcontroller, or whatever.

Here's how the programming process works on a step-by-step basis:

1. Write the source code.
2. Compile the source code into object code.
3. Link the object code with libraries to build a program.
4. Run and test the program.

Take a moment to absorb this process: write, compile, link, run. The human writes source code. The source code is compiled into object code. The object code is linked with a C library to create a program. Then, finally, that program is run.

In reality, it goes more like this:

1. Write the source code.
2. Compile the source code into object code.
3. Fix errors and repeat Steps 1 and 2.
4. Link the object code with libraries to build the program.
5. Fix errors and repeat Steps 1 through 4.
6. Run and test the program.
7. Fix bugs by repeating the entire process.

   Or, more frequently, the program runs fine but you want to add a feature or refine an element. Then you repeat everything.

Hopefully, Steps 3, 5, and 7 don't happen often. Still, you do a lot of fixing in the programming cycle.

The good news is that the computer dutifully reports the errors and even shows you where they are. That's better than tracking down a bug in miles of wires back in the old ENIAC days.

- When you use an IDE, the compile and link steps are handled by the Build command. The compile, link, and run steps are handled by the Build and Run command. Despite the term *build,* internally the IDE is still compiling object code, linking libraries, and creating a final program to run.
- One of my professional programmer friends said that the art form should be called *debugging,* not programming.
- Legend has it that the first computer bug was literally a bug that Grace Hopper found in the wiring of an early computer. There's some doubt about this legend, considering that the word *bug* has been used since Shakespeare's time to describe something quirky or odd.

TECHNICAL STUFF

Writing source code

Source code represents the part of the process that contains the programming language itself. You use a text editor to write a source code file.

In this book, source code is shown in program listings, such as the example in Listing 2-1.

Listing 2-1: Standard "Hello World" Program

```
#include <stdio.h>

int main()
{
    puts("Greetings, human.");
    return 0;
}
```

Line numbers are not shown in the listings because they can be confusing. Besides, line numbers are referenced in the editor, such as in Code::Blocks, as you type.

You're directed to type the source code from a listing as part of an exercise; for example:

Exercise 2-1: Start a new project in Code::Blocks. Name the project ex0201.

Do it: Obey Exercise 2-1 and start a new project in Code::Blocks named ex0201, according to these specific steps:

1. **Create a new Code::Blocks console application, a C language project named** ex0201**.**

   Refer to Chapter 1 if you fail utterly to understand this step.

2. **Type the code from Listing 2-1 into the editor.**

   You can erase the skeleton given by Code::Blocks or just edit it so that the result matches Listing 2-1.

3. **Save the source code file by choosing the File➪Save File command.**

There. You've just completed the first step in the programming process — writing source code. The next section continues your journey with the compiler.

- All C source code files end with the `.c` ("dot-see") filename extension.
- If you're using Windows, I recommend that you set the folder options so that filename extensions are displayed.
- C++ source code files have the extension `.cpp` ("dot-see-pee-pee"). I shall refrain from writing a puerile joke here.
- Source code files are named the same as any file on a computer. Traditionally, a small program has the same source code filename as the final program. So if your program is named `puzzle`, the source code is

named puzzle.c. The main.c filename is used by Code::Blocks, but it need not be named main.c.

- In Code::Blocks, the final program name is the same as the project name, so changing the source code filename isn't vital.

Compiling to object code

A compiler is a program that reads text from a source code file and translates that text — a programming language — into something called *object code.* In C, the compiler also deals with special instructions called *preprocessor directives.*

For example, Listing 2-1 shows the following precompiler directive:

```
#include <stdio.h>
```

The include directive instructs the compiler to locate the header file stdio.h. The contents of that file are added to the source code, and then both are converted by the compiler into object code. The object code is then saved into an *object code file.* The object file has the same name as the source code file, but with the .o ("dot-oh") filename extension.

As the compiler translates your C code into object code, it checks for common mistakes, missing items, and other issues. If anything is awry, the compiler displays a list of errors. To fix the errors, you reedit the source code and attempt to compile once again.

Continue with Exercise 2-1: In Code::Blocks, follow this step to compile:

4. **Choose Build➪Compile Current File.**

   The Build Log window displays the results, which shows zero errors and zero warnings. Well, unless you mistyped something, in which case, check your source code against Listing 2-1 in the preceding section.

You would normally choose the Build command at this step, as demonstrated in Chapter 1. But when you need only to compile, you use the Compile Current File command.

Upon success, the compiler produces an *object code file.* Because the source code file is named main.c, the object code file is named main.o.

In Code::Blocks, the object code file is found in the project's folder, inside either the obj/Release or obj/Debug subfolder.

Linking in the C library

The *linker* is the tool that creates the final program. It does so by linking the object code file with C language libraries. The libraries contain the actual instructions that tell the computer (or another device) what to do. Those instructions are selected and executed based on the shorthand directions found in the object code.

For example, in Listing 2-1, you see the word `puts`. This word is a C language function, which is written as `puts()` in this text. It stands for *put s*tring.

Oh, and *puts* rhymes with *foots,* not *shuts.*

The compiler translates `puts()` into a token and saves that token in the object code file, `main.o`.

The linker combines the object file with the C language standard library file, creating the final program. As with the compiler, if any errors are detected (primarily, unknown tokens at this point), the process stops and you're alerted to the potential troublemaker. Otherwise, a fully functional program is generated.

In Code::Blocks, the Build command is used to compile *and* link; the IDE lacks a separate Link command.

Continue with Exercise 2-1: Build the project ex0201. Follow this step:

5. **Choose the Build➪Build command.**

   Code::Blocks links the object file with C's standard library file to create a program file.

The next and final step in the process is to run the program.

REMEMBER

- The text a program manipulates is referred to as a *string,* which is any text longer than a single character. In C, a string is enclosed in double quotes:

  ```
  "Hello! I am a string."
  ```

- The final program includes the C language library, bundling it in with the object code. This combination explains why a program file is larger than the source code file that created it.
- Some C programs link in several libraries, depending on what the program does. In addition to the standard C libraries, you can link libraries for working with graphics, networking, sound, and so on. As you learn more about programming, you'll discover how to choose and link in various libraries. Chapter 24 offers the details.

Running and testing

Creating a program is the whole point of programming, so the first thing to do after linking is to run the result. Running is necessary, primarily to demonstrate that the program does what you intend and in the manner you desire.

When the program doesn't work, you have to go back and fix the code. Yes, it's entirely possible to build a program and see no errors and then find that the thing doesn't work. It happens all the time.

Continue with Exercise 2-1: Complete the process from the preceding sections, and then follow these final steps in Code::Blocks:

6. **Choose Build⇨Run.**

   The program runs. As a Text mode program, it appears in a terminal window, where you can peruse the results.

7. **Close the terminal window by pressing the Enter key on the keyboard.**

Running a simple program like ex0201 merely shows the results. For complex projects, you test the program. To do so, run the program and type various values. Try to break it! If the program survives, you've done your job. Otherwise, you have to reedit the source code to fix the problem and then rebuild the program.

- Running a program is a job for the device's processor and operating system: The *operating system* loads the program into memory, where the *processor* runs the code. That's a fairly loose description of how a program works.
- In Code::Blocks, the program file is named after the project. In Windows, the name is `ex0201.exe`. In Mac OS X, Linux, and Unix, the program name is `ex0201` with no extension. Further, in those operating systems, the file's permissions are set so that the file becomes an executable.

TECHNICAL STUFF

Code::Blocks project file accounting

Code::Blocks organizes its projects into folders. The primary folder is given the project's name, such as ex0201. Within that folder, you'll find all files associated with the project, including the source code, object code, and executable program. Here's a breakdown of what's what, assuming that the project is named ex0201:

`*.c`—The source code files. Source code files are stored in the project's main folder.

`*.cbp`—The Code::Blocks project file, named after the project. You can open this file to run Code::Blocks and work on the project.

`bin/Release/`—The folder in which the program file is stored. The file is named after the project.

`bin/Debug/`—The folder that's created when you choose a debug build target. An executable program with debugging information is stored here.

`obj/Release/`—The folder that contains object code files for the project's release version. One object code file is found for each source code file.

`obj/Debug/`—The folder that contains object code files for the debugging target.

Other files may lurk in the project's folder as well, most of them related to something Code::Blocks does. If you create more-complex projects, such as Windows programs or cell phone apps, other folders and files appear as necessary.

You have no need to probe the depths of a project folder on a regular basis, because Code::Blocks manages everything for you.